Lecture 4

Numerical solution of initial value problems

The methods you've learned so far have obtained **closed-form** solutions to initial value problems. A closed-form solution is an explicit algebriac formula that you can write down in a finite number of elementary operations. In the real world, such problems are the exception rather than the rule: most initial value problems don't have closed-form solutions. Even when a problem can be solved in closed form, the solution may be extremely difficult to obtain or just plain inconvenient to work with.

In such cases, we will need to compute approximate solutions. In very broad terms here are two kinds of approximations to consider:

- **Analytical approximations** – we compute a **formula** that approximates the true solution to the problem.
- **Numerical approximations** – we compute **numerical values** that approximate the numerical values of the true solution.

In this introductory course we'll not consider analytical approximation methods, but will have a short unit on numerical approximations.

A word about terminology: it's common to talk about "numerical solution" of differential equations. Strictly speaking, we never compute numerical solutions: we only compute numerical *approximations* to solutions. But "numerical approximation to the solution to a differential equation" is a mouthful, so we – like everyone else – will refer to "numerical solution." Just keep in mind at all times that a "numerical solution" is actually an approximation to a solution.

The idea of numerical approximation is very old, dating back (at least) to Archimedes' "method of exhaustion" for systematically refining approximations to the numerical value of π. Numerical solutions to initial value problems have been done for several centuries. In Euler's day, computations would have been done by hand with the aid of log tables. Mechanical computers ("adding machines") and finally digital computers have let numerical solutions be carried out very efficiently, and numerical methods are now one of the most widely used ways of solving differential equations.

The central challenge in numerical approximation is to find methods that produce good approximations to a true solution in a reasonable amount of work. I will lead you through an examination of the *approximation error* and then a study of how the *computational cost* depends on the desired amount of accuracy.

4.1 Approximation by Taylor Polynomials

The starting point for virtually all numerical methods for solving IVPs is Taylor's Theorem. Recall Taylor polynomials from Calculus II: to approximate a function $y(x)$ in the vicinity of a point a, form an N-th

degree polynomial

$$T_y^N(x,a) = y(a) + y'(a)\,(x-a) + \frac{1}{2}y''(a)\,(x-a)^2 + \cdots + \frac{1}{N!}y^{(N)}(a)\,(x-a)^N\,, \tag{4.1}$$

which in summation notation is written

$$T_y^N(x,a) = \sum_{k=0}^{N} \frac{1}{k!}y^{(k)}(a)\,(x-a)^k\,. \tag{4.2}$$

The rather complicated notation $T_y^N(x,a)$ means that T is an N-th degree approximation to a function $y(x)$, expanded about the point a.

Where does the Taylor polynomial come from? The N-order Taylor polynomial approximation to $y(x)$ is the unique polynomial whose first N derivatives exactly match those of $y(x)$ at the point a. Other sorts of polynomial approximations are possible: the Taylor approximation turns out to be a useful one for initial value problems, so we will concentrate on it here.

4.1.1 The error in a Taylor approximation

Except in cases where $y(x)$ is already a polynomial of degree less than or equal to N, a Taylor polynomial will never be an exact fit to y. The error in a Taylor polynomial approximation is given by the **Lagrange remainder formula**,

$$y(x) - T_y^N(x,a) = R_y^N(x,a) = \frac{1}{(N+1)!}y^{(N+1)}(c)\,(x-a)^{N+1}$$

where c is an undetermined number somewhere in the interval $[a,x]$. At first glance, the Lagrange remainder might appear to be useless because we don't know c and therefore don't know the exact value of the remainder. The importance of the Lagrange remainder is twofold: first, it tells us *roughly* how the error depends on $x-a$, the distance from the point a; second, it tells us *roughly* how the error will change as N increases. I say "roughly" because the value of c depends (in some complicated and undetermined way) on the value of x; in practice, as long as we don't wander too far from a the behavior of the remainder is dominated by the factor $(x-a)^{N+1}$.

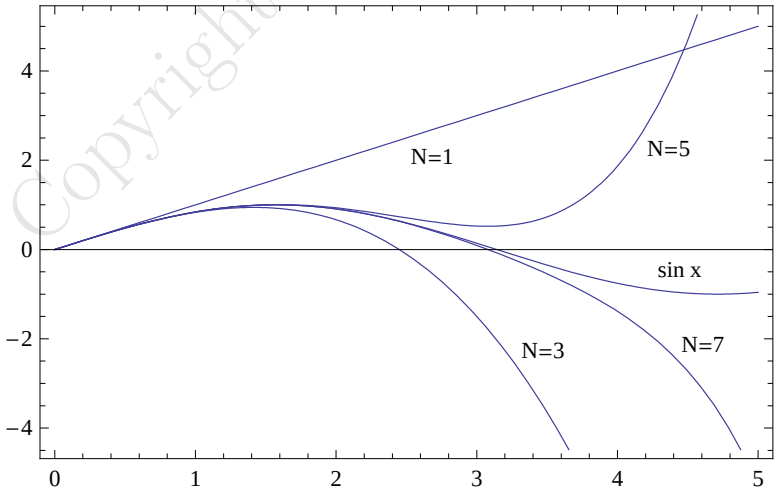

Figure 4.1: Taylor polynomial approximations to $\sin x$.

Figure 6.1.1 shows Taylor approximations to the sine function, expanded about the point $a = 0$. As the order of the approximation increases, the approximation closely follows the sine function over a greater

distance from a. Also notice that for $x \lesssim 3$, as N increases the error gets smaller: with $N = 1$ the error is large, but at $N = 7$ the polynomial approximation and the sine function are hard to distinguish by eye. Far from a, say, $x \gtrsim 3 - 4$, all of the approximations shown are wildly off. It would take a very high-order Taylor polynomial to get a good approximation to the sine function so far away from $a = 0$.

To understand this behavior, recall that the N-th order Taylor polynomial was derived by fitting the first N derivatives of y at the point a. *All of the information used in building the Taylor approximation came from the point a.* It should not be surprising that the approximation is *very* good near a, but gets *very* bad once you wander too far from that point.

Box 4.1 Approximations based on Taylor polynomials get better the smaller the "step" you take away from the point a, and this improvement is more rapid the higher the polynomial order of the approximation.

Upper bounds on the approximation error

The Lagrange remainder formula for an N-th order Taylor polynomial approximation to $y(x)$ near a is

$$R_y^N(x,a) = \frac{1}{(N+1)!} y^{(N+1)}(c)\,(x-a)^{N+1}.$$

Again, we don't know what c is and so can never compute R_y^N exactly. However, we can often put an **upper bound** on the error.

If $\left|y^{(N+1)}(x)\right|$ is bounded so that there is some maximum value ("upper bound") $M > 0$ such that

$$\left|y^{(N+1)}(x)\right| \leq M(N+1)!$$

then we can in turn put an upper bound on the Lagrange remainder:

$$\left|R_y^N\right| = \frac{1}{(N+1)!}\left|y^{(N+1)}(x)\right| |x-a|^{N+1} \tag{4.3}$$

$$\leq M\,|x-a|^{N+1}. \tag{4.4}$$

We still don't necessarily know a value for M, but we do know that as we take x closer and closer to a, the upper bound on the Lagrange remainder gets better and better. For example: if $N = 1$ and $x = a + h_1$, then

$$\left|R_y^1\right| \leq Mh_1^2.$$

If we get less adventurous and stay closer to a, say, taking a "step" one-tenth as far away from a, $x = a + \frac{1}{10}h_1$, then

$$\left|R_y^1\right| \leq \frac{1}{100}Mh_1^2.$$

If we step a tenth as far out from a, we reduce the maximum value of the Lagrange remainder by a factor of 100. Repeat this argument with $N = 3$, *i.e.* a cubic Taylor polynomial, and find that reducing the "step" by ten reduces the bound on the Lagrange remainder by 10000.

4.1.2 Order notation for error estimates

In numerical solution of IVPs we'll usually use h to represent a small "step" in the independent variable x away from the expansion point a of a Taylor polynomial. The Lagrange remainder tells us that provided $y^{(N+1)}(x)$ is bounded, there is an M such that the absolute value of the error will be less than $M\,|h|^{N+1}$. It's

cumbersome to go around saying "there exists an $M > 0$ such that the asolute value of the error is less than or equal to $|h|^{N+1}$," so we'll develop some shorthand notation for describing errors. This "order notation" is used for approximations of all sorts, both numerical and analytical.

This section contains definitions and proofs, and you might lose sight of the forest for the trees. The "bottom line" is a set of simple rules for working with error estimates, highlighted in several gray boxes.

We'll write inequalities such as

$$|E| \leq M\,|h|^p$$

for $M > 0$ in the shorthand notation

$$E = O(h^p),$$

read as "E is of order h^p." The symbol O is the letter O – often called "big O" – not the number zero. The notation $E = O(h^p)$ does *not* mean that E is given by some function O acting on h^p. We'll sometimes abbreviate even further and write $E \sim h^p$ for $E = O(h^p)$.

Box 4.2 Order notation:
$O(h^p)$ is shorthand for the statement that there exists an $M > 0$ such that the quantity E is bounded from above by $M\,|h|^p$.

When we see O-notation in an equation such as

$$a = b + O(h^3)$$

we mean that a is equal to b within an approximation error of order h^3. In other words, there is a positive number M such that

$$|a - b| \leq M\,|h|^3\,.$$

We can write the error in a Taylor approximation using this notation. For example, the cubic Maclaurin[1] approximation for $\sin x$ is

$$\sin x = x - \frac{1}{6}x^3 + O(x^5),$$

which we interpret to mean that the error in the cubic approximation to $\sin x$ is of order x^5, or again, more precisely, there exists a positive number M such that

$$\left| x - \frac{1}{6}x^3 - \sin x \right| \leq M\,|x|^5\,.$$

We'll next look at how error estimates propagate through mathematical calculations.

Multiplying an error estimate by a constant

If $E = O(h^p)$, then any constant times E is also $O(h^p)$. Consequently, when you see an expression such as

$$a = b + \frac{\pi}{137}O(h^p)$$

you can immediately rewrite it as

$$a = b + O(h^p).$$

This saves us from dragging unnecessary constants with us throughout a messy error calculation. It is similar in spirit to noticing that if C is a constant of integration, then $2\,C$ is also "just a constant."

[1] A Maclaurin approximation is a Taylor approximation with the expansion point $a = 0$.

Let's prove that this is true. By definition, $E = O(h^p)$ means that $\exists\, M > 0$ such that $|E| \leq M\,|h|^p$. If true, then for any constant A (constant in the sense of not depending on h) clearly

$$\begin{aligned} |AE| &\leq |AMh^p| && (4.5)\\ &\leq |AM|\,|h^p|\,. && (4.6) \end{aligned}$$

But $|AM|$ is just another positive constant. Call it M_2, and we have

$$|AE| \leq M_2\,|h^p|\,.$$

Therefore, there exists a constant $M_2 > 0$ such that $|AE| \leq M_2\,|h^p|$, which means $AE = O(h^p)$.

Box 4.3 Multiplicative constants don't affect order of accuracy.
If you see an expression such as

$$a = b + 42\,O(h^2),$$

you can simplify it to

$$a + b = O(h^2).$$

Errors in addition of approximate quantities

Suppose that $u_1 = \widetilde{u_1} + O(h^p)$ and $u_2 = \widetilde{u_2} + O(h^q)$. If we approximate $u_1 + u + 2$ by adding $\widetilde{u_1} + \widetilde{u_2}$, what is the error in that calculation?

The error $|E|$ in an approximate calculation $u_1 + u_2 \approx \widetilde{u_1} - \widetilde{u_2}$ will be

$$|E| = |u_1 + u_2 - \widetilde{u_1} - \widetilde{u_2}|\,.$$

By the triangle inequality,[2]

$$|E| \leq |u_1 - \widetilde{u_1}| + |u_2 - \widetilde{u_2}|\,,$$

and then because we've stipulated that $u_1 - \widetilde{u_1} = O(h^p)$ and $u_2 - \widetilde{u_2} = O(h^q)$,

$$|E| \leq M_1\,|h|^p + M_2\,|h|^q$$

where M_1 and M_2 are positive constants.

Let's arbitrarily choose $p \geq q$, or $p = q + r$ where $r \geq 0$. Let M be the larger of the two constants, $M = \max(M_1, M_2)$. Then

$$\begin{aligned} |E| &\leq M_1\,|h|^p + M_2\,|h|^q && (4.7)\\ &\leq M\,|h|^{q+r} + M\,|h|^q && (4.8)\\ &\leq M\,|h|^q\,|h|^r + M\,|h|^q && (4.9)\\ &\leq M\,|h|^q\,(1 + |h|^r) && (4.10)\\ &\leq \begin{cases} 2M\,|h|^q & h \leq 1 \\ 2M\,|h|^p & h > 1. \end{cases} && (4.11) \end{aligned}$$

In the context of Taylor approximation, we're interested in the case where $h \ll 1$, so in this context we'll always use the first case, $|E| \leq 2M\,|h|^q$. Because $2M$ is just a constant multiplier and could be "renamed" M, we have $|E| = O(h^q)$.

[2] See your calculus textbook if you don't remember the triangle inequality.

Box 4.4 When adding a quantity with error $O(h^p)$ to another quantity with error $O(h^q)$, the error in the sum is $O(h^{\min(p,q)})$. **The sum of two errors is dominated by the error with the *lower* order.**

Incidentally, in other contexts such as analysis of algorithms and asymptotic approximation, this same order notation is used to describe the behavior for *large* h. In those contexts, a sum is dominated by the term with the *larger* order.

Errors in coefficients of h^p

In the development of the Improved Euler and Runge-Kutta methods we'll encounter expressions such as

$$Ah^p$$

where A is independent of h. We will eventually want to approximate A by $A = \tilde{A} + O(h^q)$. The approximation to Ah^p is then $\tilde{A}h^p + O(h^{p+q})$. You might try to prove this statement using the definition of $O(h^q)$ in terms of error bounds.

Box 4.5 If the coefficient A of an expression Ah^p is approximated by $\widetilde{A}$ with an error $O(h^q)$, the error in $\widetilde{A}h^p$ is then $O(h^{p+q})$.

Errors in function arguments

Suppose the argument u of some function $g(u)$ is approximated to $O(h^p)$. Given that error in u, what is the approximation error in the computed value $g(u)$?

Let $\tilde{u}$ be the approximate value of u, and $E_u = |u - \tilde{u}| = O(h^p)$ be the absolute error in u. Make a zeroth-order Taylor polynomial approximation to $g(u)$ expanded about the point $\tilde{u}$, using order notation for the Lagrange remainder,

$$\begin{aligned} g(u) &= g(\tilde{u}) + O(E_u) && (4.12) \\ &= g(\tilde{u}) + O(h^p). && (4.13) \end{aligned}$$

Box 4.6 If u is approximated by $\tilde{u}$ to $O(h^p)$, then $g(\tilde{u})$ is accurate to $O(h^p)$.

4.2 Euler's Method

Euler's method is the simplest scheme for computing approximate numerical solutions to initial value problems $y' = f(x, y)$, $y(x_0) = y_0$. The idea is intuitive: near a point (x_0, y_0), approximate the solution by a line with slope $y' = f(x_0, y_0)$,

$$y(x) \approx y_0 + f(x_0, y_0)\,(x - x_0)\,.$$

Notice that because of the ODE $y' = f$, this is equivalent to

$$y(x) \approx y(x_0) + y'(x_0)\,(x - x_0)\,.$$

This is simply a first-order Taylor polynomial approximation! Now this is probably a very poor approximation to the solution to the IVP; however, refer back to figure 6.1.1 and notice that while the $N = 1$ Taylor approximation to the sine function is very poor at large x, it is quite good when $x \lesssim 0.5$. Even a low-order Taylor polynomial can be a very good *local* approximation provided we stay sufficiently close to the point x_0.

Euler's insight was to use a Taylor approximation's good behavior *near* a point x_0 while avoiding its bad behavior *far from* x_0 by making a sequence of short steps of some "small" size h rather than one big step. Let's define some notation:

- $x_n = x_0 + nh$, *i.e*, x_n is the value of x reached after n steps of size h.
- y_n is the Euler approximation to the solution $y(x)$ at $x = x_n$.

An **Euler step** from x_n to x_{n+1} is a first-order Taylor approximation to $y(x_{n+1})$ starting at the point x_n. Note that $x_{n+1} - x_n = h$, so

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{4.14}$$

Euler's method is to repeat short Euler steps until a desired point x_N is reached:

$$\begin{aligned} y_1 &= y_0 + hf(x_0, y_0) && (4.15)\\ y_2 &= y_1 + hf(x_1, y_1) && (4.16)\\ y_3 &= y_2 + hf(x_2, y_2) && (4.17)\\ &\vdots && (4.18)\\ y_N &= y_{N-1} + hf(x_{N-1}, y_{N-1}) && (4.19) \end{aligned}$$

4.2.1 The local error

We can use the Lagrange remainder formula to estimate the error in the first Euler step: recalling that $x_1 - x_0 = h$, the error will be

$$\frac{1}{2!}y''(c)h^2$$

or simply $O(h^2)$.

This is the error in the first step. The next step to x_2 will have a similar error, *plus an additional error resulting from having started the second step at an approximately-computed* y_1. The step to x_3 will then have its own error, plus the error resulting from starting at a value of y_2 that's itself based on two approximate steps. If you're incredibly lucky – which you almost *never* will be – all of those errors will cancel. In the more typical case we need to figure out how all those single-step errors accumulate.

We'll call the error in a single step the **local error**. The error accumulated after many steps will be called the **global error**, which we examine next.

4.2.2 The global error

Let's ignore for the moment the "compounding" effect of building each step on previous steps that are themselves only approximate, and look at the effect of taking N steps, each with an error $O(h^2)$, to traverse an interval $[x_0, x_N]$. To cover that interval with steps of size h, you must take $N = \frac{x_N - x_0}{h}$ steps. If each step has error $O(h^2)$, the error after N steps will be

$$\frac{x_N - x_0}{h}O(h^2) = O(h).$$

Thus, the global error is one power of h worse than the local error.

The effect of compounding is more difficult to analyze, but it can be shown that the effect of compounding approximations leads to the same conclusion as above: to get the global error, reduce the local error's exponent on h by one. For Euler's method, the local error $O(h^2)$ becomes a global error $O(h)$. A method with $O(h^5)$ local error becomes $O(h^4)$ globally, and so on.

4.2.3 Computational cost of numerical solution

Suppose you have used a stepsize h_1 to compute a numerical solution with absolute global error $E(h_1)$. You can use this information to estimate the stepsize h_2 needed to reach a desired error $E(h_2)$. Because $E \leq Mh^p$ with a p-th order numerical solution method, the *worst-case* error of a p-th order method with stepsize h is $E(h) = Mh^p$. Then

$$E(h_1) = Mh_1^p \tag{4.20}$$
$$E(h_2) = Mh_2^p \tag{4.21}$$

and after eliminating the unknown M and solving for h_2,

$$h_2 = \left(\frac{E(h_2)}{E(h_1)}\right)^{1/p} h_1.$$

Thus, to reduce the error by a factor $R = \frac{E_2}{E_1}$ (not to be confused with the letter R as used in the Lagrange remainder; think R for "reduction" in this context) you must change the stepsize from h_1 to $h_2 = h_1 R^{1/p}$.

The number of steps of size h required to traverse an interval of fixed size L will be $N = L/h$, so if you have taken N_1 steps of size h_1, you will need $N_2 = \frac{h_1}{h_2} N_1$ steps of size h_2 to traverse the same interval. Combining this with the previous result relating change in stepsize to desired change in error, we find

$$N_2 = \frac{N_1}{R^{1/p}}.$$

Now, in making this argument I assumed that we were always at the *worst-case* error, so that the inequality $E \leq Mh^p$ could be replaced by an equality $E = Mh^p$. In practice, we won't necessarily hit the worst-case error, so the true equalities don't hold. It's best to think of the formulas we've derived as approximate equalities:

$$h_2 \approx \left(\frac{E(h_2)}{E(h_1)}\right)^{1/p} h_1$$

and

$$N_2 \approx \frac{N_1}{R^{1/p}}.$$

At this point, the following clever idea might occur to you: if we were assuming everything was a worst-case error, if we take

$$N_2 = \frac{N_1}{R^{1/p}}$$

steps, won't we always do *better* than the desired error reduction R? The answer is no. The reason is that we assumed that *both* $E(h_1)$ and $E(h_2)$ used the worst-case M. Suppose it happened that for some problem a calculation with stepsize h_1 gave a *better* approximation than the worst case possible with that stepsize. Then, going to a smaller stepsize might not help as much as theoretically possible.

These approximate rules for stepsize reduction will become more reliable as the stepsize is reduced.

Box 4.7 To reduce the absolute global error in a p-th order approximation by a factor R, reduce the stepsize by a factor

$$\frac{h_2}{h_1} \approx R^{1/p}.$$

This will require a factor

$$\frac{N_2}{N_1} \approx \frac{1}{R^{1/p}}$$

more steps. These approximate rules for stepsize reduction will become more reliable – *i.e* closer to equalities – as the stepsize is reduced.

Example

To reduce an error $E_1 = 0.01$ to an error $E_2 = 10^{-6} = 10^{-4}E_1$ we need a reduction factor $R = 10^{-4}$. Euler's method is first-order, so it will take

$$N_2 \approx \frac{N_1}{R} = 10^4 N_1$$

steps to achieve the desired reduction.

Suppose you have found a method with global error $O(h^2)$. Then, to improve the error by $R = 10^{-4}$ takes

$$N_2 \approx \frac{N_1}{\sqrt{R}} = 10^2 N_1$$

steps.

4.3 Improving Euler's method

The **improved Euler method** is a simple extension of Euler's method giving second-order accuracy. The obvious way to get higher order of accuracy would be to take more terms in the Taylor approximation. Euler's method used the first-order term; why not try including the second-order terms as well? Then we would have

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{1}{2}y''(x_n)h^2 + O(h^3)$$

But what is $y''(x_n)$? With $y' = f(x, y)$ and the multivariable chain rule, we can find

$$y'' = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}y'$$

and so

$$y''(x_n) = \left.\frac{\partial f}{\partial x}\right|_{x_n, y_n} + \left.\frac{\partial f}{\partial y} \cdot f\right|_{x_n, y_n}.$$

This is somewhat complicated to compute, and will be even more so when there is more than one unknown function or when an even higher-order method is desired.

The second derivative is hard to compute, but do we really need it? Suppose you can *approximate* the second derivative y'' by some $\widetilde{y''(x_n)}$, accurate to $O(h^q)$:

$$y''(x_n) = \widetilde{y''(x_n)} + O(h^q)$$

Then

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2}\left(\widetilde{y''(x_n)} + O(h^q)\right) + O(h^3).$$

To what accuracy do we need to approximate the second derivative? Recall from 6.1.2 that if we approximate A by $\widetilde{A}$ with accuracy $O(h^q)$, then $h^p A$ is accurate to $O(h^{p+q})$. So, upon replacing y'' by its q-th order approximation $\widetilde{y''}$, we have

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2}\widetilde{y''(x_n)} + O(h^{2+q}) + O(h^3).$$

The first error term, $O(h^{2+q})$, is the result of approximating y'' by $\widetilde{y''}$. The second, $O(h^3)$, is the result of chopping the initial Taylor series after the second-derivative term. Using the rule for adding errors, the combined local error in y_{n+1} will be

$$O(h^{\min(3,2+q)}).$$

This tells us the accuracy requirement for the estimate $\widetilde{y''}$: if we want to do better than Euler's $O(h^2)$ local error, we need $q \geq 1$ so that we get $2 + q \geq 3$. Furthermore, there's no point to getting y'' *more* accurately

than $q = 1$, because the overall error would still be $O(h^3)$ just from the error of the original second-order Taylor polynomial. Therefore, we want $q = 1$.

So while it might be hard to compute y'', the good news is that we don't actually need it! Any *first-order* approximation to y'' will contribute only $O(h^3)$ to the overall local error in the step, same as the error from stopping at second order terms to begin with.

4.3.1 Estimating y''

How do we estimate y''? It's easier than you might think. In calculus, you learned that the derivative of a function $g(x)$ is the limit of a difference quotient

$$g'(x) = \lim_{h->0} \frac{g(x+h) - g(x)}{h}.$$

It seems reasonable to approximate g' by skipping the passage to the limit and simply using the difference quotient

$$\widetilde{g'(x)} = \frac{g(x+h) - g(x)}{h}.$$

How accurate is that approximation? As always, use Taylor's theorem: $g(x+h) = g(x) + g'(x)h + O(h^2)$, so

$$\widetilde{g'(x)} = \frac{g(x) + g'(x)h + O(h^2) - g(x)}{h} \tag{4.22}$$

$$\widetilde{g'(x)} = g'(x) + O(h). \tag{4.23}$$

In other words, the difference between g' and $\widetilde{g'}$ is $O(h)$, which is exactly the accuracy needed for our purposes.

To approximate y'' to $O(h)$, form a difference quotient

$$\widetilde{y''(x_n)} = \frac{y'(x_{n+1}) - y'(x_n)}{h}.$$

But $y'(x) = f(x, y)$, so

$$\widetilde{y''(x_n)} = \frac{f(x_{n+1}, y_{n+1}) - f(x_n, y_n)}{h}.$$

Putting this approximate expression for y'' into the Taylor polynomial for y_{n+1} gives us

$$y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2} \frac{f(x_{n+1}, y_{n+1}) - f(x_n, y_n)}{h} + O(h^3) \tag{4.24}$$

$$= y_n + \frac{h}{2} \left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right] + O(h^3). \tag{4.25}$$

I've already absorbed the error in $h^2 y''$ into the overall $O(h^3)$ error.

4.3.2 Estimating $f(x_{n+1}, y_{n+1})$

There's still one hitch: the unknown y_{n+1} appears on the RHS of the formula for the step, as indicated with a box:

$$y_{n+1} = y_n + \frac{h}{2} \left[f(x_n, y_n) + f(x_{n+1}, \boxed{y_{n+1}}) \right] + O(h^3)$$

In principle, we could solve (algebraically) for y_{n+1}, and in fact for certain problems that's a good approach. However, if we keep in mind that this whole calculation is already only an approximation, why not use an *approximation* for the quantity in the box rather than the exact value of y_{n+1}. Let's call that approximation

q_{n+1}, and say that it's accurate to $O(h^r)$. Then, by the rule in 6.1.2 for the error in a function value given error in the function's argument, we get

$$f(x_{n+1}, y_{n+1}) = f(x_{n+1}, q_{n+1}) + O(h^r).$$

This approximation to f arises in an expression that is a coefficient of h, so by section 6.1.2 we know it will contribute $O(h^{1+r})$ to the overall error. We want $O(h^3)$ overall, so we must have $r = 2$ meaning q_{n+1} must be a *second-order* approximation to y_{n+1}.

Luckily, we already know a way to get a second-order approximation to y_{n+1}: Euler's method! A single Euler step to y_{n+1} has local error $O(h^2)$. Let's see how to put all this together.

4.3.3 The improved Euler method

The approximation developed in the preceding section defines an **improved Euler step** from y_n to y_{n+1}. We can encapsulate it as a procedure:

Box 4.8 Improved Euler step:

1. Make a single Euler step to compute q_{n+1}, a first approximation to y_{n+1}:

$$q_{n+1} = y_n + hf(x_n, y_n).$$

2. Advance the independent variable: $x_{n+1} = x_n + h$.

3. Compute the *average* of the initial slope $f(x_n, y_n)$ and the slope at the (approximate) new point, $f(x_{n+1}, q_{n+1})$:

$$s_{avg} = \frac{1}{2}\left[f(x_n, y_n) + f(x_{n+1}, q_{n+1})\right].$$

4. Make a step from (x_n, y_n) to (x_{n+1}, y_{n+1}) using s_{avg} as the slope:

$$y_{n+1} = y_n + hs_{avg}$$

The resulting y_{n+1} will be accurate to a local error $O(h^3)$.

As with Euler's method, the procedure is repeated for many steps. When doing multiple steps, errors will accumulate so that the global error becomes $O(h^2)$.

4.4 Summary

1. A **Taylor polynomial** approximates a function near a point a. The coefficients in the Taylor polynomial are chosen so that its first N derivatives match the first N derivatives of the function at a. Taylor polynomials can be *very* good approximations near a, but are increasingly poor approximations the farther you go from a.

2. **Order notation** keeps track of the dependence of an error estimate on an approximation parameter h. The notation $E = O(h^p)$ reads "E is order h^p," or more briefly, "E is p-th order." Saying "x is accurate to p-th order" is equivalent to saying "the error in x is p-th order."

Box 4.9 Rules for working with error estimates:

(a) $O(h^p)$ is shorthand for the statement that there exists an $M > 0$ such that the quantity E is bounded from above by Mh^p.

(b) Multiplicative constants don't affect order of accuracy.

(c) When adding a quantity with error $O(h^p)$ to another quantity with error $O(h^q)$, the error in the sum is $O(h^{\min(p,q)})$.

(d) If the coefficient A of an expression Ah^p is approximated by $\widetilde{A}$ with an error $O(h^q)$, the error in $\widetilde{A}h^p$ is then $O(h^{p+q})$.

(e) If u is approximated by $\tilde{u}$ to $O(h^p)$, then $g(\tilde{u})$ is accurate to $O(h^p)$.

3. **Euler's method** advances from (x_n, y_n) to (x_n+h, y_{n+1}) using a first-order Taylor polynomial approximation. One then computes the first derivative at the new point, and advances from there. Because it is based on a first-order polynomial, the **local error** in a single step is $O(h^2)$. Because errors accumulate from step to step and because the number of steps required is inversely proportional to h, the **global error** is $O(h)$.

4. In principle, we could get an approximation of higher order of accuracy than Euler's method by retaining more terms in the Taylor polynomial used at each step; in practice, however, the computation of the additional derivatives appearing in the higher-order terms is difficult. The insight behind the **improved Euler method** is that we don't need to compute the derivatives appearing in those higher-order terms exactly: we need only approximate them to an accuracy sufficient to keep the total error in the step the same order as the error in the Taylor polynomial approximation.

   An improved Euler step approximates the second derivative $y''(x_n)$ by a difference quotient having $O(h)$ accuracy. When computing the step, the second derivative appears only in a term $h^2y''(x_n)$, so the $O(h)$ error in the approximate y'' results in a $O(h^3)$ local error, exactly the same order as the error from truncating the Taylor polynomial.

   In computing the difference quotient approximation to y'', the $f(x_{n+1}, y_{n+1})$ appears, and then h times this quantity appears in the formula for the step to y_{n+1}. But y_{n+1} isn't yet known until we take the step! However, if we approximate y_{n+1} to $O(h^2)$ accuracy by some q_{n+1}, then $hf(x_{n+1}, q_{n+1})$ will be accurate to $O(h^3)$, no worse than the previous approximations. We compute q_{n+1} using a step of Euler's method, with $O(h^2)$ local error.

5. If a method is known to have $O(h^p)$ global error, we can use that information to estimate the stepsize needed to reduce the error by a given factor. To reduce the error by a factor R, the rule of thumb is to reduce the stepsize by $R^{1/p}$. This in turn increases the cost of the computation by a factor $1/R^{1/p}$.